

Brief intro to Large Language Models

Contents

- Recall the main points of LLMs on surface level
- Recall neural networks
- Transformers
- Using LLMs

Recall the main points of LLM on the surface level

LLM (Large Language Model / Foundation model) like GPT: what does it do?

Try to find the next word for the sentences:

They are leaving in about fifteen ...

Dog ate my ...

The Panama-registered tanker was carrying 136,000 tons of ...

North Korea has agreed to send a delegation to next month's Winter Olympics
in ...

Officials from the two nations met ...

$2 + 2 = \dots$

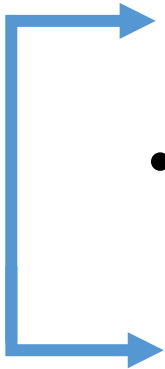
cat => kass, dog => ..

What does it do? Predict tokens (letters, words) one at a time:

- John sat in the car and ...
- John sat in the car and started ...
- John sat in the car and started to ...
- John sat in the car and started to drive ...
- John sat in the car and started to drive home ...

How to predict? A simplified idea:

- Collect statistics from a lot of texts: how did the sentences continue
John sat in the car and ... (*started, thought, ...*)
- When you see a sentence you have not seen before, use similar ones:
Mike sat on a bike and ...



Really large LLMs?

GPT4 (March 2023)

- 16 separate models (experts)
- Each expert has ca 111 billion params (ca 222 GB of memory?)
- Trained on ca 13 TB of text (ca 3 million large books)?
- OpenAI spent more than \$100 million training

LLM like any neural network is an interpreted computer program

- Parameters (small floating point numbers) are the “code” .
- An **interpreter program** reads the code and produces results.
- Example interpreter programs: llama.cpp, mistral.rs, ...

Differently from “conventional program”, neural networks are built by **training on examples**.

How can I run / use an LLM?

- **Open / free** ones like Llama from Meta:
 - On your own laptop (cpu: very slow; gpu: needed 6 GB ... 200 GB).
 - Some Taltech server with a gpu with a large video memory.
- **Large / proprietary** like GPT or Claude: on their servers either via
 - Web page for human use.
 - API for calling from command line or your own program.

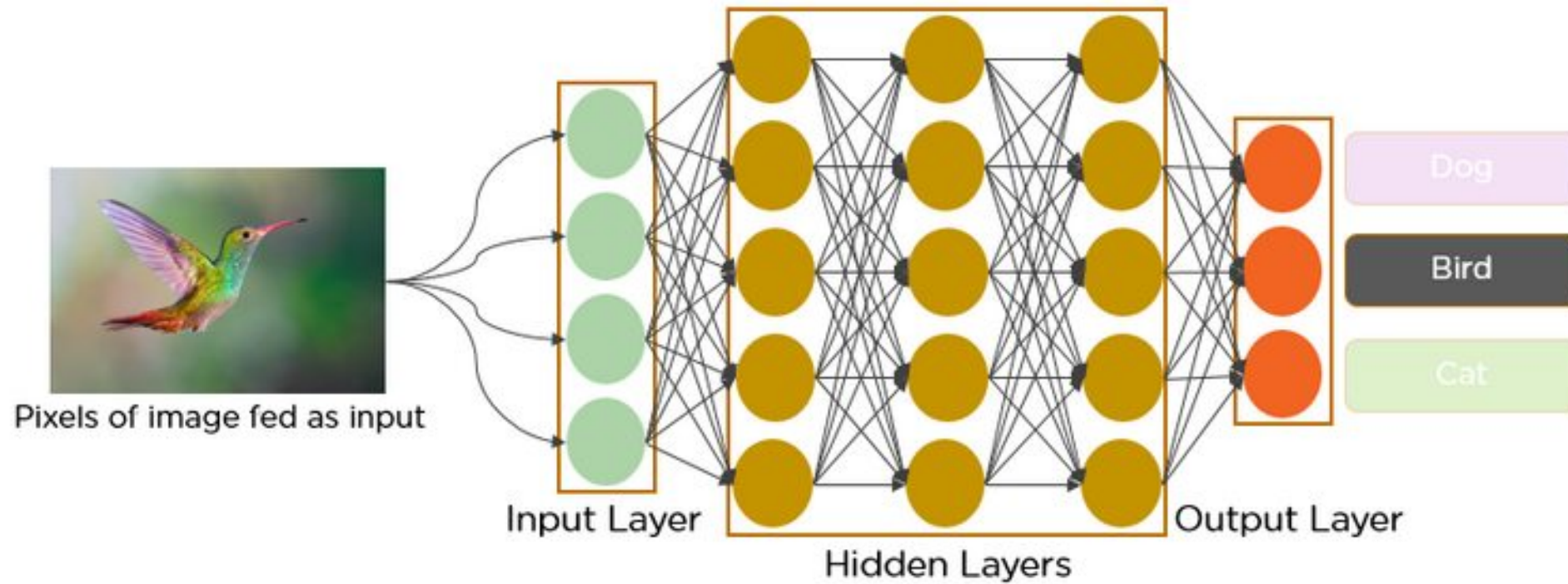
“Large Language Model” aka LLM ?

- Given text (prompt), **predicts next word**, then next, then next, ...
- Is a **foundation** model.
- Built using the **transformer** architecture.
- **Large amounts of compute** have been used for training
- Not necessarily “very large” (like, Phi-3-mini: 3.8 billion params, 4 bits per parameter, ca 1.8 GB of memory; trained on 3.3 TB of words / tokens)

Recall neural networks

	Common AI Techniques					
Use Case Families	Generative Models	Non-Generative ML	Optimisation	Simulation	Rules	Graphs
Forecasting	Low	High	Low	High	Medium	Low
Planning	Low	Low	High	Medium	Medium	High
Decision Intelligence	Low	Medium	High	High	High	Medium
Autonomous System	Low	Medium	High	Medium	Medium	Low
Segmentation	Medium	High	Low	Low	High	High
Recommender	Medium	High	Medium	Low	Medium	High
Perception	Medium	High	Low	Low	Low	Low
Intelligent Automation	Medium	High	Low	Low	High	Medium
Anomaly Detection	Medium	High	Low	Medium	Medium	High
Content Generation	High	Low	Low	High	Low	Low
Chatbots	High	High	Low	Low	Medium	High
Knowledge Discovery	High	Medium	Low	Low	Medium	High

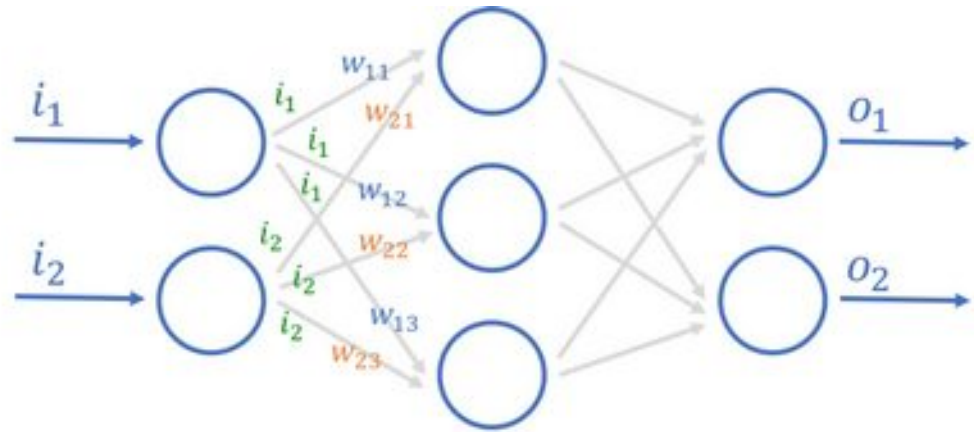
Deep neural networks is the most popular M.L. technique among many



Matrices and calculations

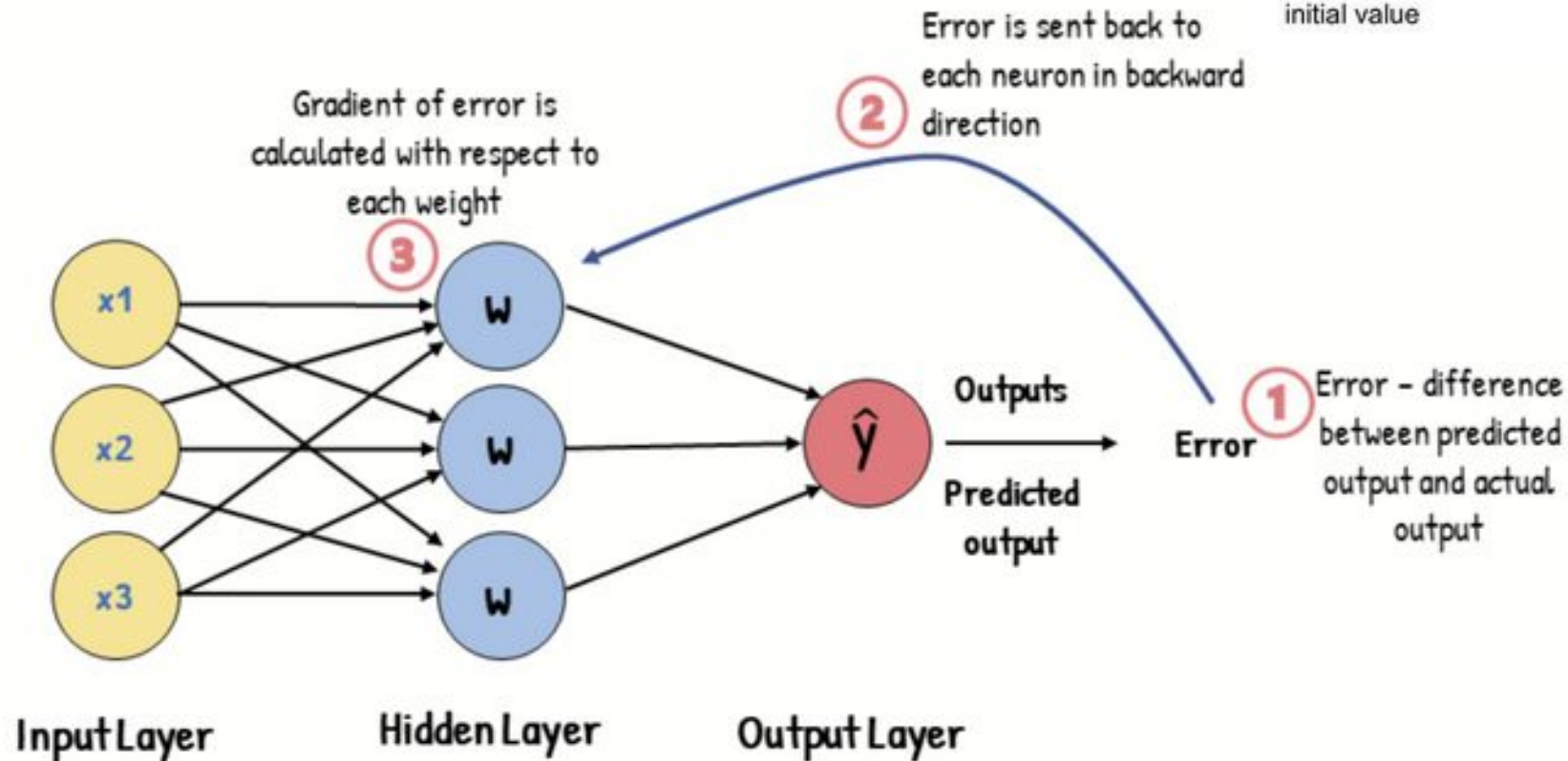
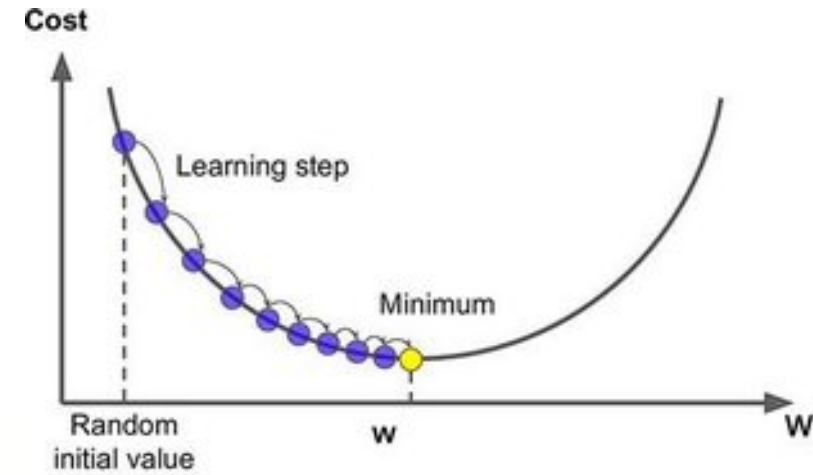
Connections are simply numbers (weights) you can store in a large table (matrix).

Calculating values for nodes can be done in several ways. One simple approach is to first multiply weights with inputs, sum all of them together and then normalize, using some popular simple function for this.



$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix}$$

Learning by backpropagation and gradient descent



Stuff to read / look at

SLP3 slides about simple neural networks for text:

https://web.stanford.edu/~jurafsky/slp3/slides/7_NN_Apr_28_2021.pdf

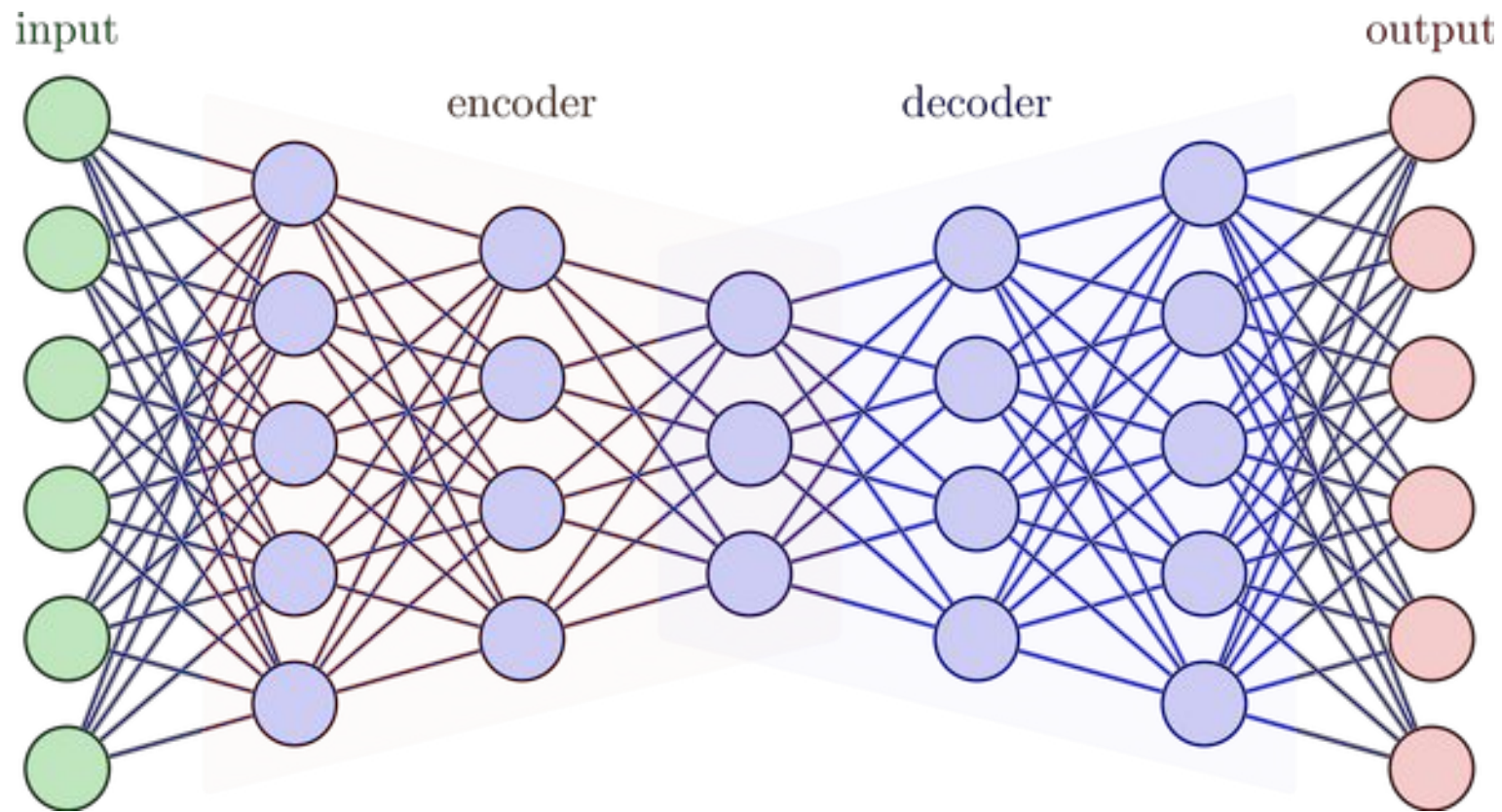
Gradient descent:

<https://medium.com/data-science/gradient-descent-algorithm-a-deep-dive-cf04e8115f21>

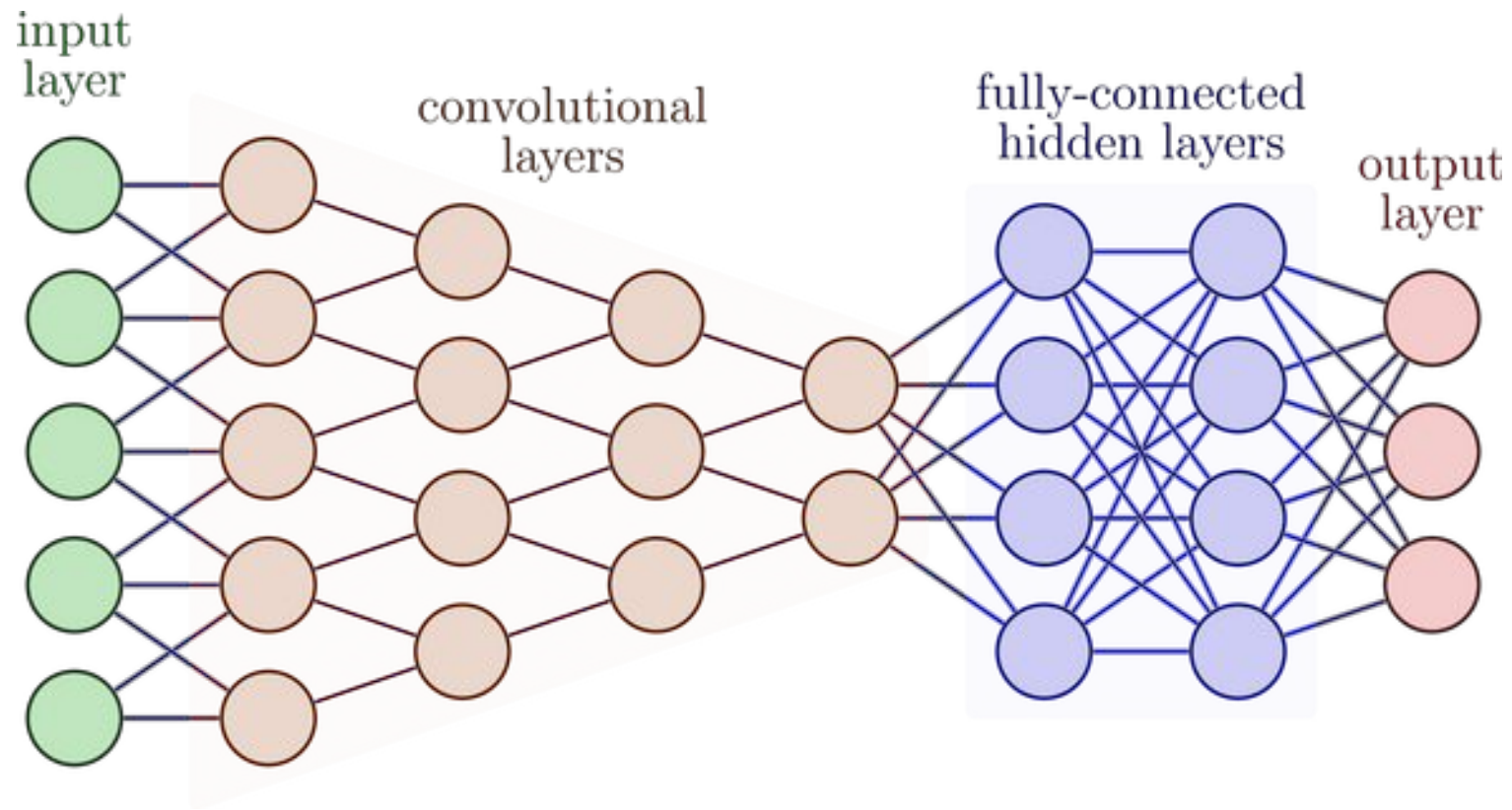
2003 paper of next word prediction with simple neural networks:

<https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf>

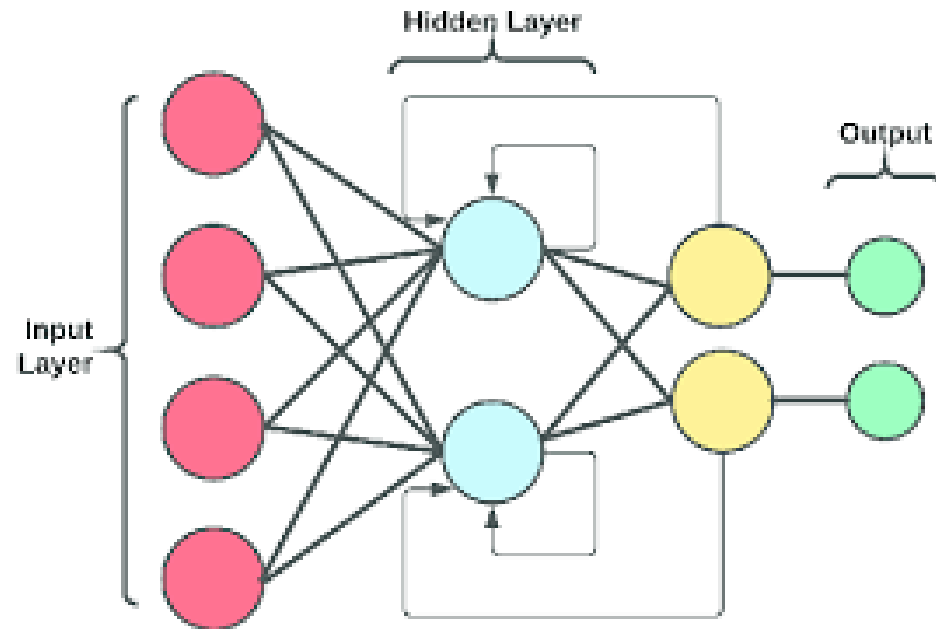
Autoencoder neural network (denoising, compression, generation)



Convolutional neural network (image recognition)



Recurrent neural network (time series, etc)



Transformers

Transformer (large language models
aka LLMs like BERT / GPT / Llama)

Originally developed by Google in 2017
for machine translation. BERT model
uses both the encoder and decoder.

GPT uses only the decoder part.

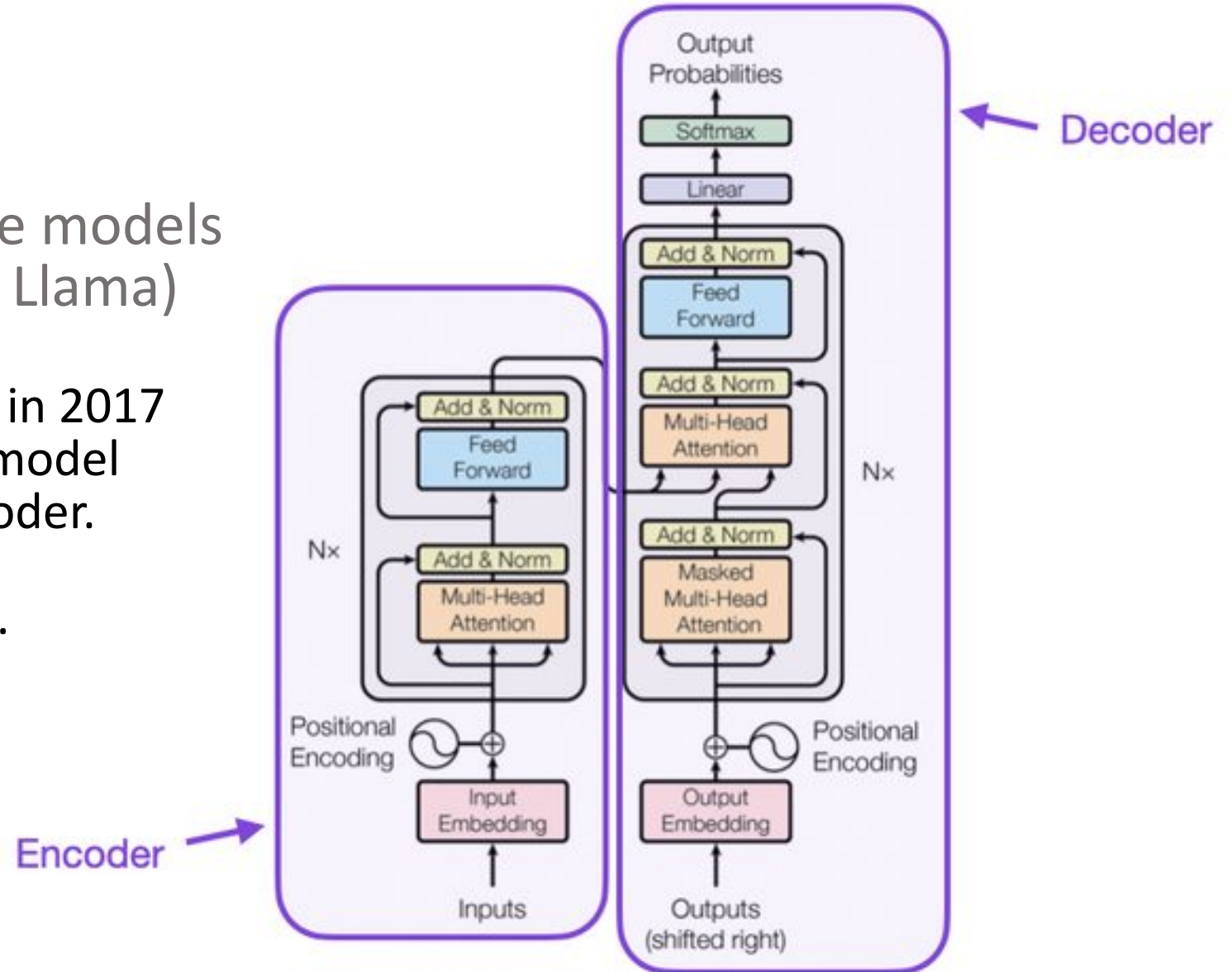


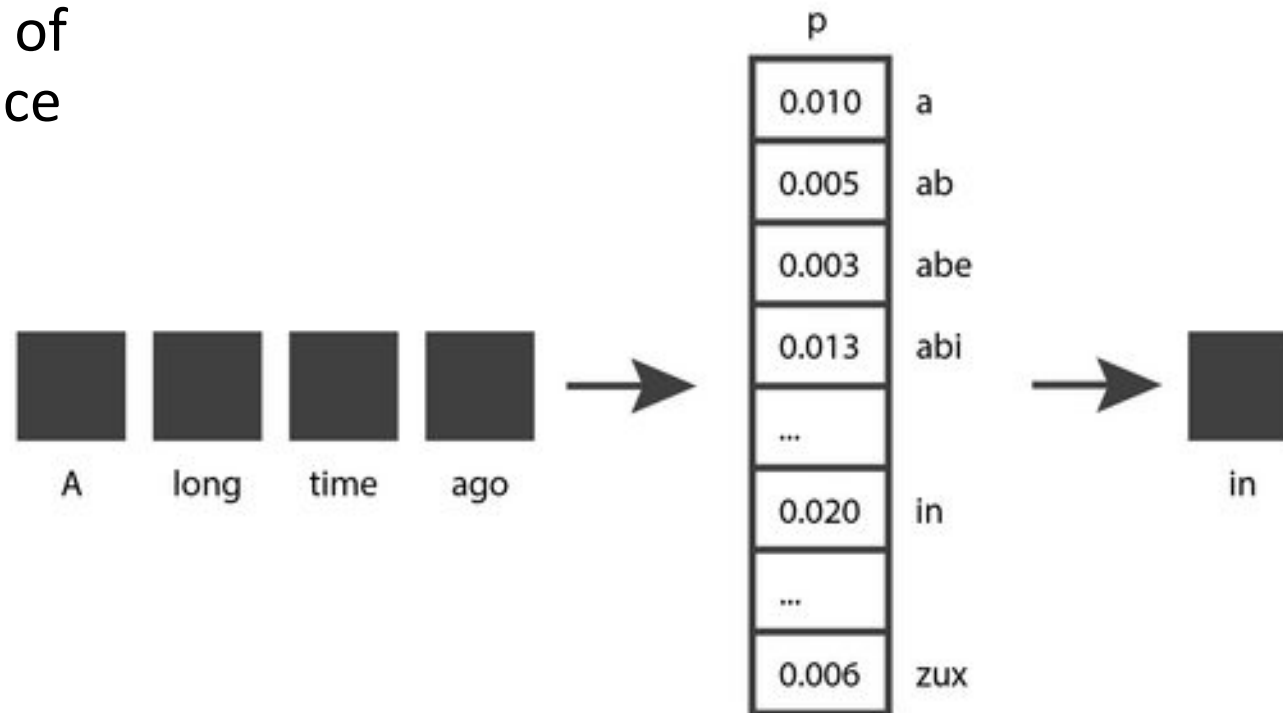
Figure 1: The Transformer - model architecture.

Known about GPT3 (predecessor of GPT4)

- Number of layers: 96
- Number of attention heads: 96
- Dimensions of its hidden layers: 12288
- Sequence length: 2048
- Number of parameters: 175 billion (ca 350 GB)

LLMs: predict next word / token

Each token / word has an approximate probability of being next in the sentence



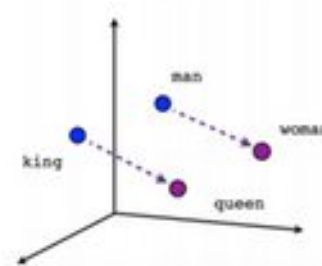
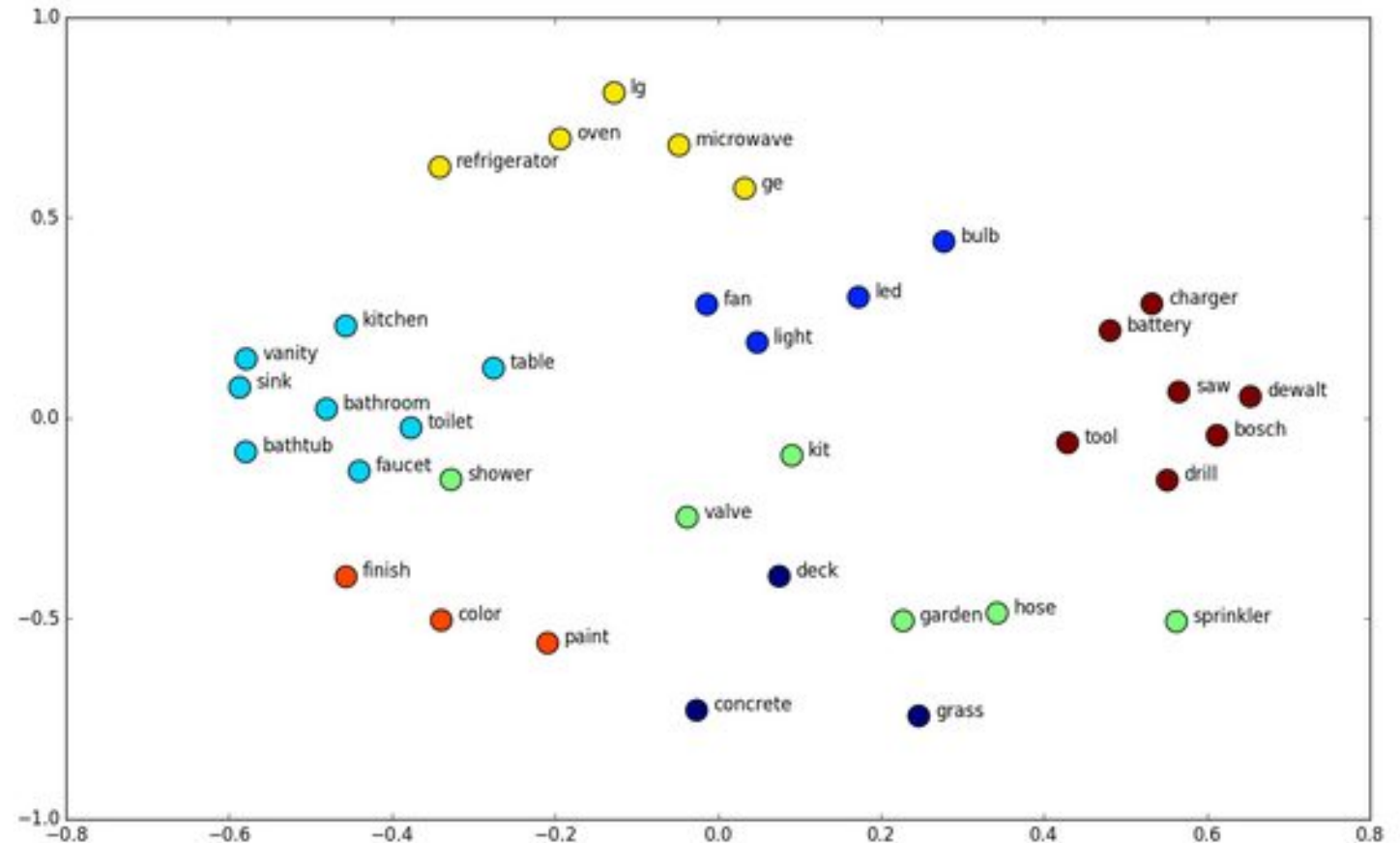
Word vectors (embeddings)

Each word or a phrase like “car”, “horse”, “jumped”, “fast runner”, “weird colour of sky” is represented by a long list of numbers like

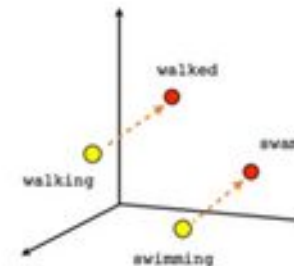
1, 99, 23, ..., 85,12

with each number indicating the “closeness” of the word/phrase to some specific concept or property.

GPT4 has ca 3000 numbers in a word vector.



Male-Female

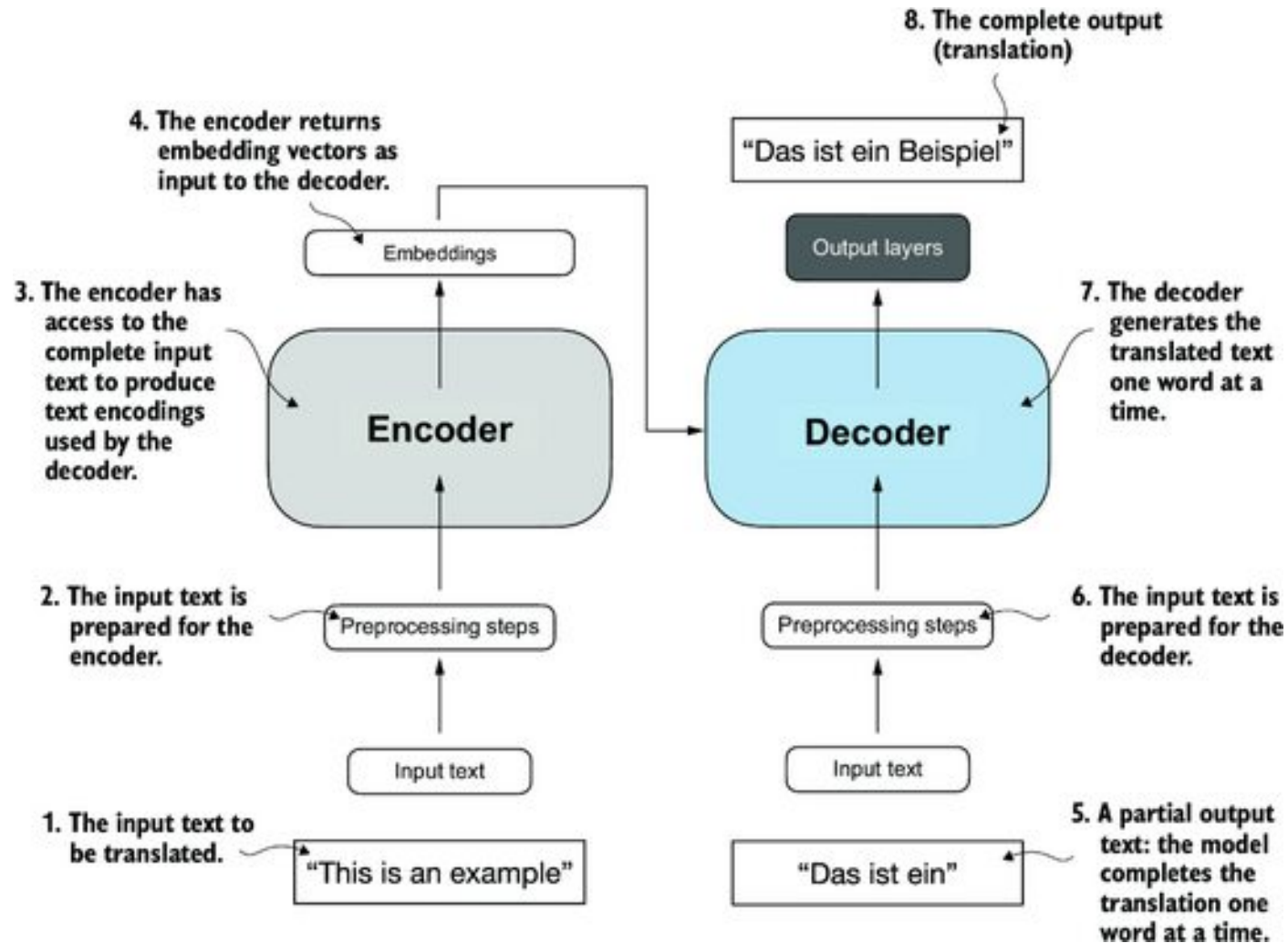


Verb tense

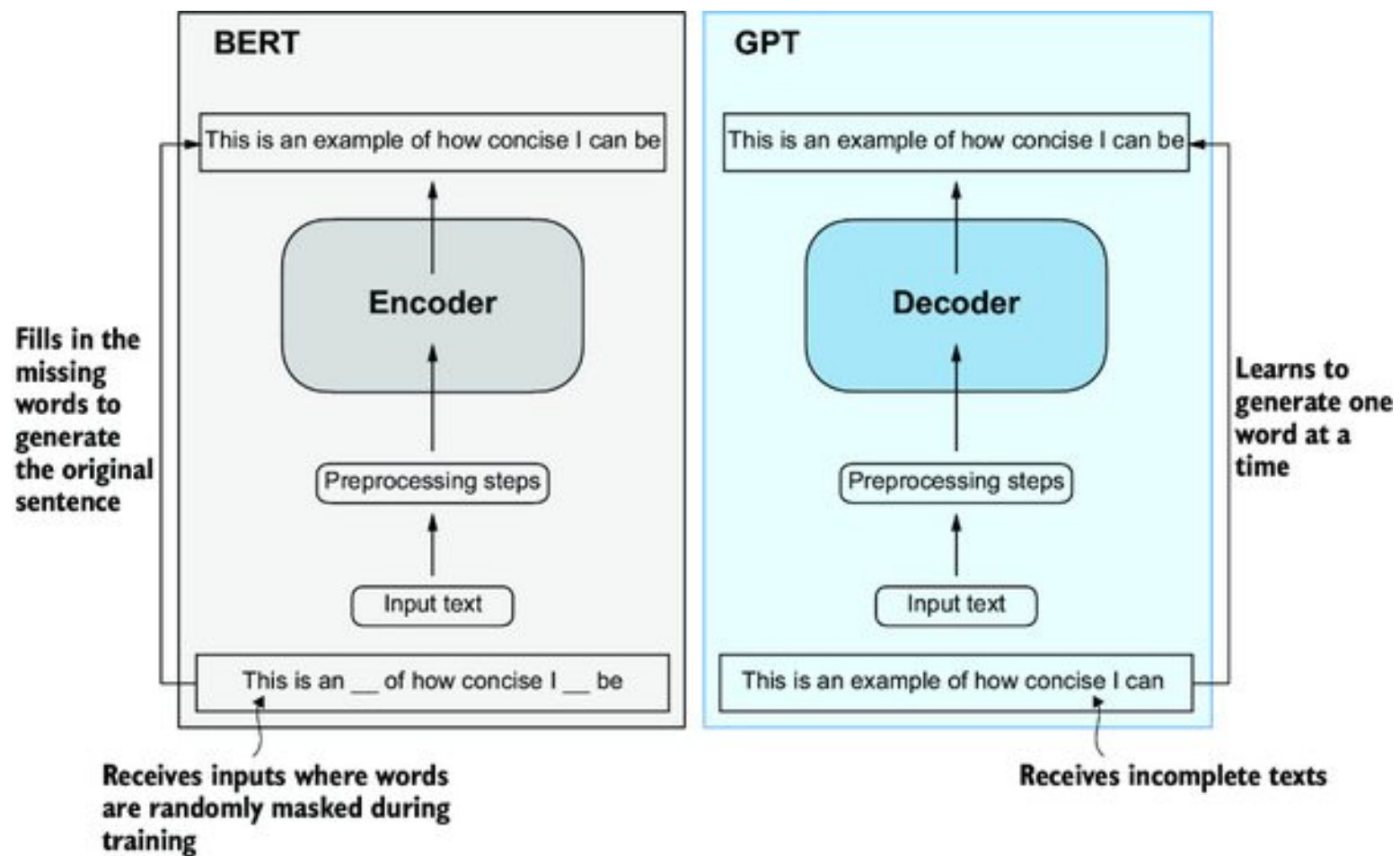


Country-Capital

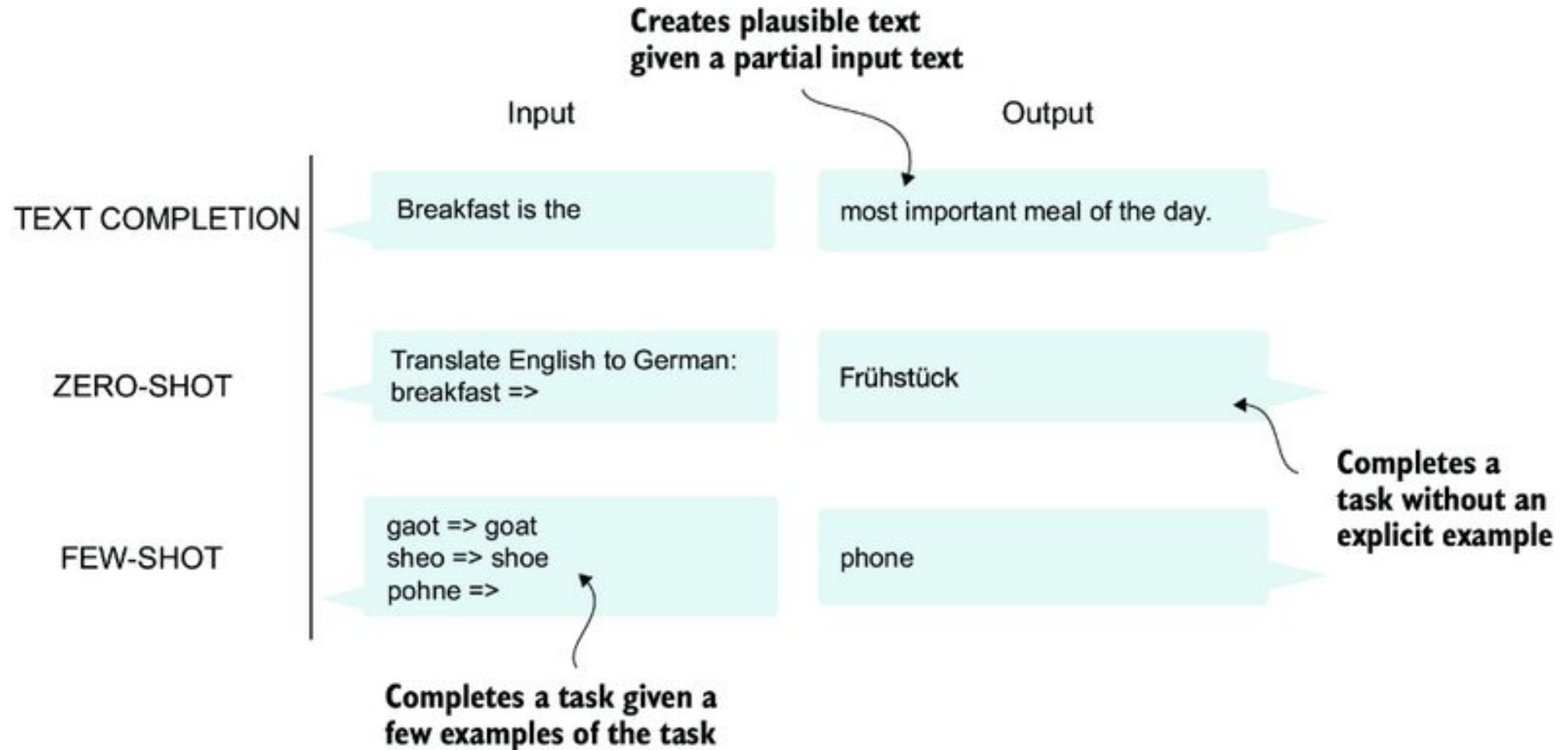
Encoder and decoder (original architecture for translation)



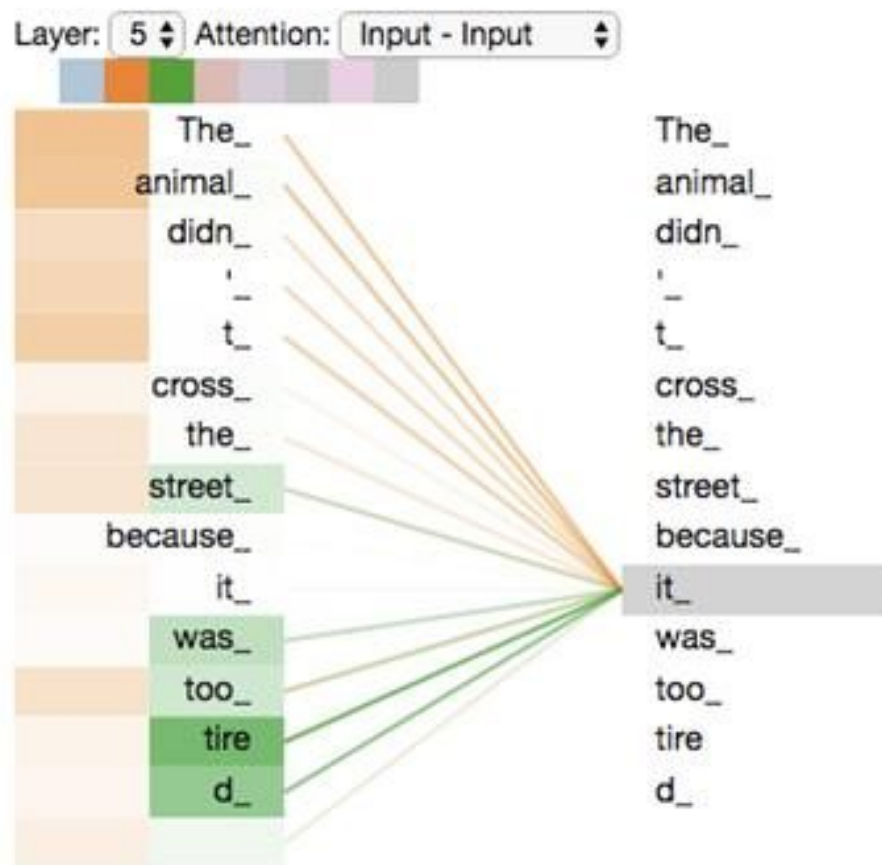
BERT vs GPT



Shots (examples)



Attention



As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

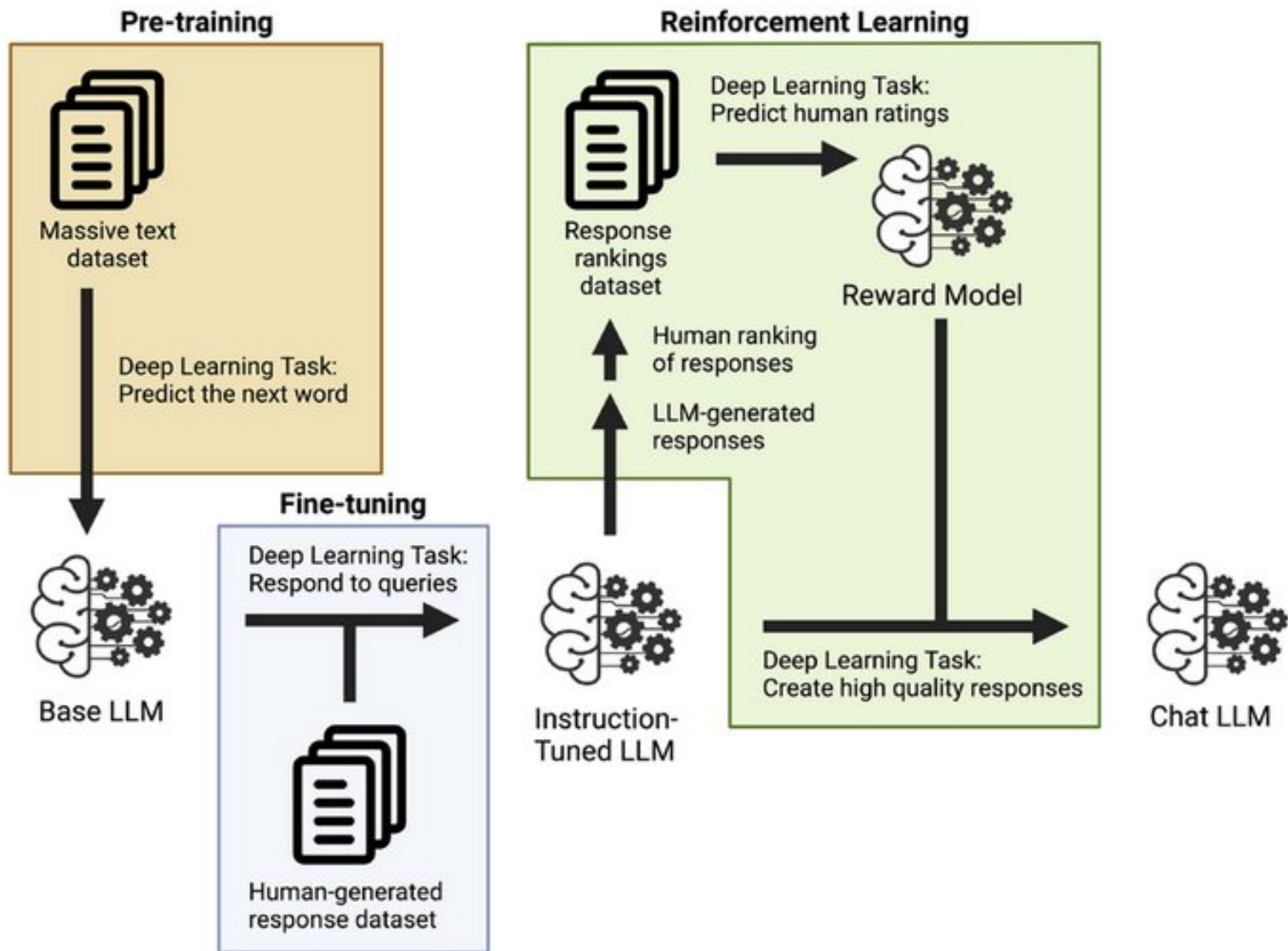
Attention as transforming information of word vectors: example for GPT2 (predecessor of GPT3)

When Mary and John went to the store, John gave a drink to ... GPT2 predicts: Mary.

Three types of attention heads contributed to this prediction:

- Three heads copied information from the Mary vector to the final input vector (for the word to). GPT-2 uses the information in this rightmost vector to predict the next word.
- How did the network decide Mary was the right word to copy? A group of four attention heads marked the second John vector in a way that blocked the first three heads from copying the name John.
- How did the second head group decide John shouldn't be copied? Two attention heads marked the second John vector as a duplicate of the first John vector, which helped the second head group to decide that John shouldn't be copied.

These nine attention heads enabled GPT-2 to figure out that “John gave a drink to John” doesn't make sense and choose “John gave a drink to Mary” instead



Stuff to read / look at

The illustrated transformer:

<https://jalammar.github.io/illustrated-transformer/>

Using LLMs

Reasoning: partially OK

- LLMs perform **limited reasoning**.
- Nobody knows where exactly is the plateau or limit.
- Deep / uncommon reasoning fails, though.
- LLMs **do not contain search / planning / simulation algorithms**.

Recent paper by Apple research: “adding seemingly relevant but ultimately inconsequential information to the logical reasoning of the problem led to substantial performance drops of up to 65% across all state-of-the-art models. “

Our experiments with *GPT 4o* for small NL-to-logic tasks

Problem set	Set size	LLM err	Json err 1	Json err 2	Prove err
hans	113	15%	8%	4%	12%
core	982	18%	19%	2%	28%
allen	38	18%	34%	18%	42%
wiki	90	2%	40%	10%	49%

The lawyers were recommended by the doctor. The lawyers recommended the doctor?

Crenkils are not birds. John is a crenkil. John is a bird?

John had a car Eve liked. Eve liked a car? Eve liked the car? Eve had a car?

The case of machine learning: gaining new knowledge by machine learning a software system to predict answers or actions similar to what has been observed

Issues:

- A very large number of examples is necessary, as well as a lot of compute
- Errors are inevitable; more generalization leads to more errors
- LLMs: hallucination (errors presented with confidence)
- Inability to explain the reasons for answers given
- Inability to perform search, for example, nontrivial planning or finding proofs
- Inability to efficiently incorporate software tools
- No memory: learning new information is very hard; forgetting is even harder

Issue: hallucinations (confidently generating wrong answers)

Human: Who was the mother of Afonso II, the third king of Portugal?

ChatGPT: The mother of Afonso II was Queen Urraca of Castile

Comment: *totally invented wrong “fact”*

Issue: hallucinations (confidently generating wrong answers)

Human: how James Joyce and Vladimir Lenin first met?

ChatGPT: James Joyce and Vladimir Lenin met in Zurich, Switzerland in 1916. Both men were living in exile in Zurich during World War I. Joyce was a writer and Lenin was a revolutionary. They met at the Cafe Odéon, a popular gathering place for artists and intellectuals in Zurich.

Comment: an encounter of these people has never been confirmed

Issue: not much of a reasoner (unless you use a well-known example)

Human: A man and a goat are on one side of the river. They have a boat. How can they both go across?

ChatGPT: The man takes the goat across the river first, leaving the boat on the original side. Then, the man returns alone with the boat. He leaves the goat on the other side and takes the boat back to the original side. Finally, he takes the cabbage across the river.

Issue: no short-term memory, very hard to teach anything, extremely hard to forget

Does not know anything about your specific context / company / organization

Two main approaches to alleviate:

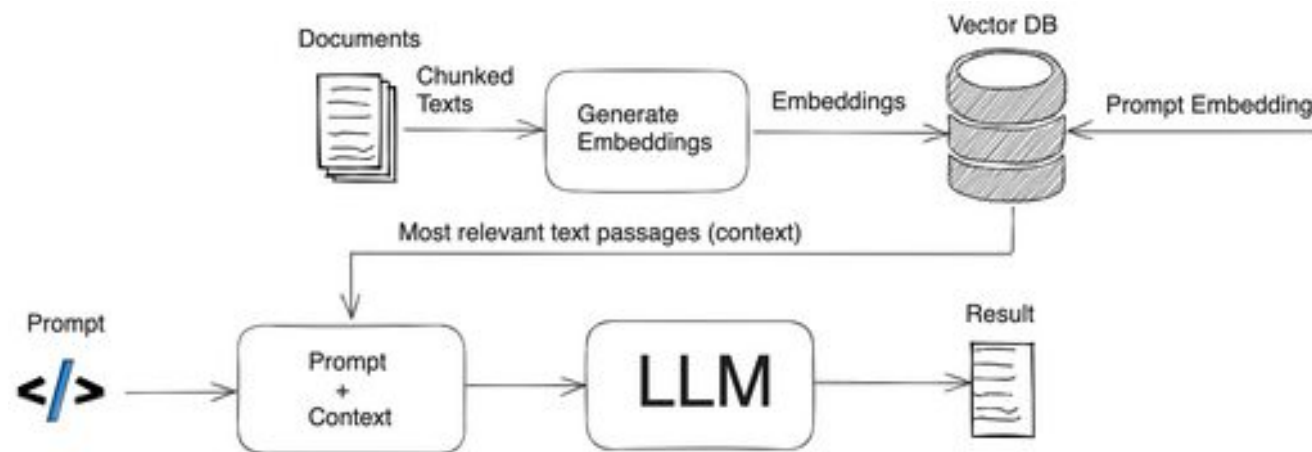
- **Retrieval-Augmented Generation aka RAG**: use special software to find relevant snippets from the docs of your context / organization and add these automatically to the prompt.
- **Fine-tuning**: perform additional teaching (like 1000 examples or so) to specialize for your context.

RAG (retrieval augmented generation): the **main** principle of using LLMs in an organization

Idea: search for relevant text snippets and add these to the LLM prompt.

The standard process with specialized RAG software:

- Collect all **potentially relevant texts** of an application domain and **index** all their short sections/paragraphs.
- Upon seeing a question, **automatically select these sections/paragraphs** which seem to be relevant. Use vector similarities / knowledge graphs and final evaluation.
- Automatically **add likely relevant text snippets** to the question prompt as background.



LLM + RAG based on automatic search

Check out <https://perplexity.ai>

“When you ask Perplexity a question, it uses advanced AI [to search the internet in real time](#), gathering insights from top-tier sources. It then distills this information into a clear, concise summary, delivering exactly what you need in an easy-to-understand, conversational tone.”

- Can find up-to-date sources not used for GPT training
- Experience is like Google mixed with GPT
- Since it uses an LLM (GPT / Gemini / ...) it still hallucinates, though less than GPT
- Probably cannot find the documents / materials of your company, but ...
- Enterprise version allows uploading your own text, pdfs etc to be used during search for relevant background

Another use case of LLMs: word / phrase vectors for RAG and similar

- Give GPT API a word, sentence or a large piece of text.
- Ask for the vector aka embedding (ca 3000 floating point numbers) for this.
- Suppose you have stored a large number of texts along with vectors.

Then you can, for example, use the vector of the question to find your existing texts / phrases with a similar vector.

General notes about LLMs for writing text

- Do not blindly trust that what it writes is correct.
- Look out for uncommon words and overuse of adverbs and remove those (“delve” etc).
- LLMs do not search for new information from web: they stay at the point they were trained.
- They are superhumanly good at imitating style. Try out with instructions and examples.
- They have a tunable randomness factor plus additional inherent randomness: the results are not too well repeatable.
- Use the top-level paid versions: better than free versions and cheap for common use cases.

Recommendations for specialized use cases

- A large prompt is better than a small prompt, up to a degree.
- **Give context:** explain briefly what role the LLM should take and what you expect.
- **Give examples:** even a few examples help, but tens / a hundred help even more.
- Explicitly prohibit the behaviour / output you observe but do not want.
- For reasoning-requiring tasks, it helps to instruct like “Reason step by step”.
- The longer the output, the more likely that LLM goes astray / makes errors.

An example of a prompt with a lot of instructions

You preprocess sentences for a semantic parser to logic. **Simplify**, maximally shorten and split the sentence to shortest possible separate subsentences, to make it understandable for children.

Replace figurative phrases like 'the price of X is above' or 'the cost of X is over' with more precise phrases like 'the price of X is more than'.

Do not use complex phrases or grammatical constructions: try to use the simplest and most common grammatical constructions and phrases possible.

If the original sentences contains verbs like 'use' with an object with a strong semantic relation in the same sentence, do not remove that object.

Do not remove adjectives like 'big', 'green' etc.

Do not introduce new pronouns like 'it', 'he', 'they': use the names and words originally present in the sentence to refer to these objects.

An example prompt with ca 30 examples covering the typical / important / nontrivial cases

You preprocess sentences for a semantic parser to logic and have to solve **coreference resolution**. Rewrite input sentences by solving the coreference resolution in the following manner:

- * To each pronoun (it, they, this, etc) append the noun phrases present in the sentences this pronoun stands for, putting curly braces around them like {Mary} or {John Smith, Mary}.
- * If a noun refers to another noun phrase in the in the sentences, append the other noun phrase to the current one, putting curly braces around them like {Steve} or {elephant, young tiger}.

Do not make any other changes in the input text, except adding the phrases in curly braces!

Examples:

Mary was in a room. She was in the room? => Mary was in a room. She {Mary} was in the room?

If an animal has a trunk, it is an elephant. => If an animal has a trunk, it {an animal} is an elephant.

If John is in a box, he is in the house. => If John is in a box, he {John} is in the house.

A baby was in a bed. She liked the place. => A baby was in a bed. She {A baby} liked the place {a bed}.

A spade is a tool for digging. It is often made of metal. => A spade is a tool for digging. It {A spade} is often made of metal.

An elephant was strong. The animal lifted a stone. => An elephant was strong. The animal {An elephant} lifted a stone.

John, Mike and a small elephant went to swim. They were happy. => John, Mike and a small elephant went to swim.

They {John, Mike, a small elephant} were happy.

...

John criticized Bill. He lost the laptop. => John criticized Bill. He {Bill} lost the laptop.

A major question:

How far can pure M.L. – and LLMs – go without explicit symbolic components?

Nobody knows: views from one end of spectrum to another

A major related question:

Do LLMs build a human-like (or some kind of) world model inside?

Again, nobody knows.

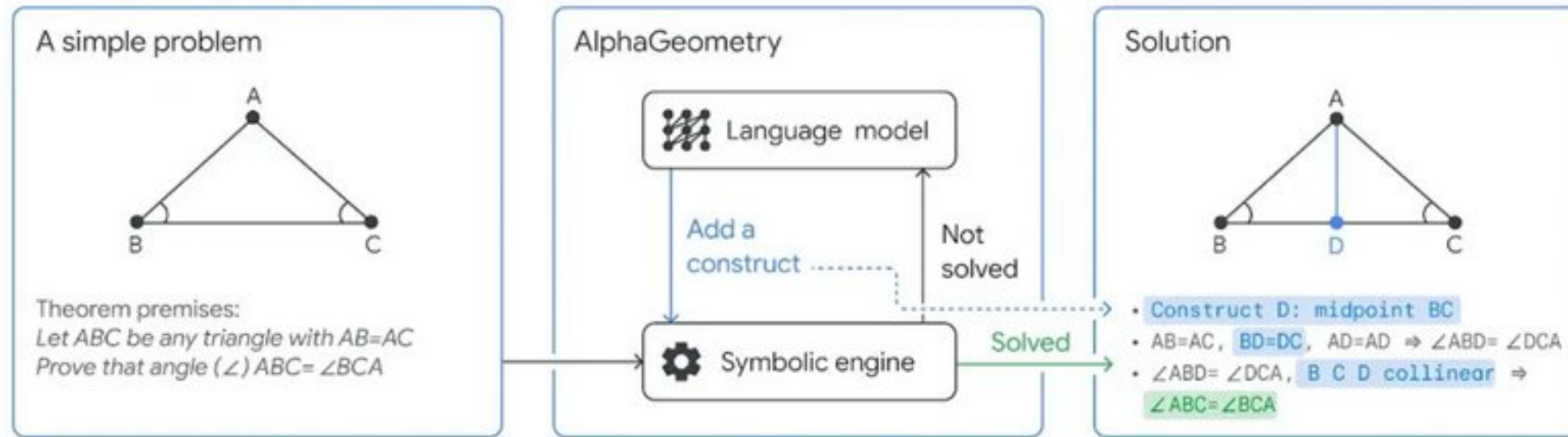
Can we help LLMs to build an adequate world model by adding video to learning?

Neurosymbolic A.I. aka hybrid A.I.: a research area for combining symbolic and ML methods

Some notes:

- An active research area, not an established applications toolkit
- A huge number of different possible approaches
- Existing practical A.I. systems are mostly already neurosymbolic: autonomous driving, leading chess engines, math proof systems, google search, alexa & siri, successes at geometry etc

Google DeepMind AlphaGeometry: autoformalization + symbolic reasoning engine



Solves complex geometry problems at a level approaching a high-school Olympiad gold-medalist

Research area of neurosymbolic A.I.:

Find ways to combine M.L. with symbolic components to be able to solve different really hard tasks.

Some approaches:

- Use M.L. to heuristically guide proof search for symbolic reasoning
- Build LLMs capable of autoformalization and then use symbolic reasoning
- Let LLMs learn on symbolic knowledge in addition to free text
- Build new architectures for M.L. capable of learning external software tool use
- Build new architectures for M.L. capable of performing search / planning
- Build symbolic rules already into a neural switch level of M.L. systems